

A Primal-Dual Algorithm for a Heterogeneous Traveling Salesman Problem

Jungyun Bae¹, Sivakumar Rathinam²

Abstract

Surveillance applications require a collection of heterogeneous vehicles to visit a set of targets. In this article, we consider a fundamental routing problem that arises in these applications involving two vehicles. Specifically, we consider a routing problem where there are two heterogeneous vehicles that start from distinct initial locations, and a set of targets. The objective is to find a tour for each vehicle such that each of the targets is visited at least once by a vehicle and the sum of the distances traveled by the vehicles is a minimum. We present a primal-dual algorithm for a variant of this routing problem that provides an approximation ratio of 2.

Keywords

Approximation algorithms, Primal-Dual method, Traveling Salesman Problem, Heterogeneous vehicles, Prize collecting TSP

INTRODUCTION

Heterogeneous unmanned vehicles are commonly used in surveillance applications for monitoring and tracking a set of targets. For example, in the Cooperative Operations in Urban Terrain project [1] at the Air Force Research Laboratory, a team of unmanned vehicles are required to monitor a set of targets and send information/video about the targets to the ground station controlled by an human operator. The human operator may further add new locations of potential targets or task the vehicles to revisit the targets at different angles. Once the human operator enters his/her input through the human-machine interface, the central computer associated with the interface has few minutes to determine the motion plans for each of the vehicles. A fundamental subproblem that has to be solved by this computer is the problem of finding a tour for each vehicle so that each target is visited at least once by some vehicle and an objective that depends on the distances traveled by the vehicles is a minimum. A common objective that is used for these applications is the sum of the total distances traveled by all the vehicles. If there is only one vehicle, this routing problem is referred to as the Traveling Salesman Problem (TSP) in the literature. If there are multiple vehicles that (possibly) start from different initial locations or depots, then this routing problem is referred to as the Multiple Depot, TSP. Once the routing problem is solved and the tours have been determined, a nominal trajectory can be specified for each vehicle to include other kinematic constraints of the vehicles using the results in [2], [3].

A multiple depot, TSP is a generalization of the single TSP and is NP-Hard. This routing problem is further complicated if the vehicles involved are heterogeneous. In this article, vehicles are considered to be heterogeneous if the distance to travel between any two targets depend on the type of the vehicle used. In the context of unmanned applications, as a multiple depot heterogeneous TSP is generally a subproblem that needs to be solved, we are interested in developing fast algorithms that produce approximate solutions than find optimal solutions that may be relatively difficult to solve. In addition, the cost of traveling between any two targets may not be accurately known because the exact locations of the targets may not be accurately known, or because the wind conditions used to determine the travel cost could change. For these reasons, the main focus of this article is to develop approximation algorithms for

1. Graduate Student, Department of Mechanical Engineering, Texas A & M University, College Station, Texas, U.S.A 77843.
2. Assistant Professor, Department of Mechanical Engineering, Texas A & M University, College Station, Texas, U.S.A 77843.

heterogeneous TSPs. An approximation algorithm for a problem is an algorithm that runs in polynomial time and produces a solution whose cost is at most a given factor away from the optimal cost for every instance of the problem.

The objective of this article is to develop a primal-dual algorithm for a two depot, heterogeneous TSP (2DHTSP). In addition to assuming that the costs satisfy the triangle inequality for each vehicle, we consider a variant of the problem where the cost of traveling between any two targets for the first vehicle is at most equal to the cost of traveling between the same targets for the second vehicle. Using these assumptions, we show that the developed primal-dual algorithm has an approximation ratio of 2. We are motivated to address this variant of the 2DHTSP due to the following reasons:

- The 2DHTSP considered in this article is one the simplest cases of the general multiple depot, heterogeneous TSP. The objective of this work is to first develop good algorithms that can handle these simpler cases efficiently.
- Consider a scenario where each of the vehicles is modeled as a ground robot that can move both forwards and backwards with a constraint on its minimum turning radius. If the minimum turning radius of the first vehicle is at most equal to the minimum turning radius of the second vehicle, it follows that the optimal distance required to travel between any two targets for the first vehicle will be at most equal to the optimal distance required for the second vehicle. Therefore, the problem addressed in this article is a useful variant to address.
- The 2DHTSP considered in this article is a generalization of a 2 depot, homogeneous TSP where there are additional *vehicle-target* constraints which require one of the vehicles to necessarily visit a given subset of targets in addition to visiting any common target available for both the vehicles. This variant of 2 depot, homogeneous TSP arises in applications where the distance to travel between the targets are identical for both the vehicles, but one of the vehicles carry sensors that require the vehicle to visit a subset of targets compulsorily.
- For some missions involving identical vehicles, it is sometimes necessary to minimize the maximum cost incurred by any of the vehicles. This problem is referred to as the min-max, multiple depot, homogeneous TSP in the literature. If there are only two vehicles involved, an application of Lagrangian duality transforms this min-max problem to the variant of heterogeneous TSP considered in this article.

Without the assumptions on the costs of the two vehicles, the 2DHTSP is a generalization of the standard variant of the prize collecting TSP considered by Goemans and Williamson in [10]. In this variant, each target essentially has a penalty associated with it. The objective of the prize collecting TSP is to find a tour for the vehicle that starts and ends at the depot such that the cost of the tour plus the sum of the penalties of each target not present in the tour is a minimum. For any two vertices i and j , if π_i, π_j denote the penalties of i and j respectively, then one can pose the prize collecting TSP as a 2DHTSP by setting the cost of traveling the edge joining vertices i and j for the second vehicle to be equal to $\frac{\pi_i + \pi_j}{2}$. Essentially, by choosing the penalty variable corresponding to the second depot to be equal to 0, one can deduce that the travel cost for the second vehicle is actually equal to the sum of the penalties of the targets not present in the tour of the first vehicle. Even though there are no penalties explicitly mentioned in the 2DHTSP, the tour cost for the second vehicle which essentially account for targets not visited by the first vehicle act as penalties. For these reasons, the primal-dual algorithm presented in this article is related to the primal-dual algorithm available for the prize-collecting TSP in [10]. Essentially, our algorithm is based on the well known moat growing procedure proposed by Goemans and Williamson in [10].

Most of the work in the literature related to approximation algorithms for multiple depot, TSPs deal with identical vehicles. For example, when the costs satisfy the triangle inequality, there are several approximation algorithms for the multiple depot, homogeneous TSP in [3]-[6]. Recently, a 3-approximation algorithm was presented for a two depot,

heterogeneous TSP in [7]. This algorithm partitions the targets by solving a linear programming relaxation and then uses Christofides algorithm [8] to find a sequence of targets for each vehicle.

The 2-approximation algorithms available in the literature for the multiple depot, TSP generally follow a two-step procedure. In the first step, a constrained forest problem which is generally a relaxation of the multiple depot, TSP is solved optimally. In the second step, an Eulerian graph is found for each vehicle based on the constrained forest. From these Eulerian graphs, a tour can be found for each vehicle by short-cutting any target already visited by a vehicle. In this article, we follow a similar procedure where we first find a heterogeneous spanning forest using a primal-dual algorithm by solving a relaxation of the 2DHTSP. Then, the edges in the heterogeneous spanning forest are doubled to obtain an Eulerian graph for each vehicle. Given these Eulerian graphs, one can [9] always find a tour for each vehicle that visits each of the targets exactly once. *The crux of this procedure depends on finding a good heterogeneous spanning forest.* Using a primal-dual algorithm, we find a heterogeneous spanning forest whose cost is at most equal to the optimal cost of the 2DHTSP in polynomial time. Hence, it follows that the approximation ratio of the proposed procedure is 2.

I. PROBLEM STATEMENT

Let $D = \{d_1, d_2\}$ denote the two depots (initial locations) corresponding to the first and the second vehicle respectively. Let T be the set of targets to be visited by both the vehicles. As there are two heterogeneous vehicles, we use two edges to connect any two targets such that each edge is used by one of the vehicles. Let $V_1 := T \cup \{d_1\}$ be the set of vertices corresponding to the first vehicle and $V_2 := T \cup \{d_2\}$ be the set of vertices corresponding to the second vehicle. Let E_1 denote the set of all the edges used by the first vehicle that join any two vertices in V_1 . Similarly, let E_2 represent the set of all the edges used by the second vehicle that join any two vertices in V_2 . Note that $E_1 \cap E_2 = \emptyset$. Given any two vertices i, j , we use $\{i, j\}_1 \in E_1$ to represent the edge used by the first vehicle to travel between i and j . Similarly, we use $\{i, j\}_2 \in E_2$ to represent the edge used by the second vehicle to travel between i and j .

Let the cost of traversing an edge $e \in E_1$ for the first vehicle be denoted by $cost_e^1$. Similarly, let the cost of traversing an edge $e \in E_2$ for the second vehicle be denoted by $cost_e^2$. We will assume that it is always cheaper to travel between any two targets using the first vehicle as compared to using the second vehicle, *i.e.*, for any two targets $i, j \in T$, $cost_{\{i,j\}_1}^1 \leq cost_{\{i,j\}_2}^2$. We also assume that the costs satisfy the triangle inequality for both the vehicles, *i.e.*, for $m = 1, 2$, for every $i, j, k \in V_m$, $e_1 := \{i, j\}_m, e_2 := \{j, k\}_m, e_3 := \{i, k\}_m$, $cost_{e_1}^m + cost_{e_2}^m \geq cost_{e_3}^m$.

A tour for the first vehicle starts from its depot d_1 , visits a set of targets in a sequence and finally returns to d_1 . A tour for the second vehicle starts from its depot d_2 , visits a set of targets in a sequence and finally returns to d_2 . Note that a tour for the first vehicle uses edges only from E_1 and a tour for the second vehicle uses edges only from E_2 . The objective of the 2DHTSP is to find a tour for each vehicle such that each target is visited exactly once by some vehicle and the sum of the cost of the edges traveled by both the vehicles is a minimum.

II. PROBLEM FORMULATION

The 2DHTSP can be formulated as an integer linear program as follows:

$$\min \sum_{e \in E_1} cost_e^1 x_e + \sum_{e \in E_2} cost_e^2 y_e \quad (1)$$

$$\sum_{e \in \delta_1(S)} x_e + 2 \sum_{T \supseteq U \supseteq S} z_U \geq 2 \quad \forall S \subseteq T, \quad (2)$$

$$\sum_{e \in \delta_2(S)} y_e \geq 2 \sum_{T \supseteq U \supseteq S} z_U \quad \forall S \subseteq T, \quad (3)$$

$$\sum_{e \in \delta_1(u)} x_e + 2 \sum_{T \supseteq U \supseteq \{u\}} z_U = 2 \quad \forall u \in T, \quad (4)$$

$$\sum_{e \in \delta_2(u)} y_e = 2 \sum_{T \supseteq U \supseteq \{u\}} z_U \quad \forall u \in T, \quad (5)$$

$$\sum_{U \subseteq T} z_U \leq 1, \quad (6)$$

$$x_e \in \{0, 1\} \text{ for } e = \{u, v\}_1 \quad \forall u, v \in T,$$

$$y_e \in \{0, 1\} \text{ for } e = \{u, v\}_2, \forall u, v \in T,$$

$$x_e \in \{0, 1, 2\} \text{ for } e = \{d_1, v\}_1 \quad \forall v \in T,$$

$$y_e \in \{0, 1, 2\} \text{ for } e = \{d_2, v\}_2, \forall v \in T,$$

$$z_U \in \{0, 1\} \quad \forall U \subseteq T. \quad (7)$$

In the above formulation, x_e is the integer variable that represents whether edge $e \in E_1$ is present in the tour corresponding to the first vehicle. For any edge e joining *two targets*, x_e can take values only in the set $\{0, 1\}$; $x_e = 1$ if e is present in the tour of the first vehicle and $x_e = 0$ otherwise. We have formulated the problem such that any of the vehicles can choose to visit just one target if required. Therefore, for any edge e joining the depot d_1 and a target $v \in T$, x_e can choose any of the values in the set $\{0, 1, 2\}$. Similarly, y_e is the integer variable that represents whether edge $e \in E_2$ is present in the tour corresponding to the second vehicle. z_U is a binary variable that determines the partition of targets connected to the first and the second depot; z_U is equal to 1 if each target in $U \subseteq T$ is connected to the second depot using edges from E_2 and each target in $T \setminus U$ is connected to the first depot using edges from E_1 . Equation (6) ensures that z_U can be equal to 1 for at most one subset of T .

In equations (2,3), $\delta_i(S)$ (for $i = 1, 2$) denotes the subset of all the edges of E_i with one end in S and an other end in $V_i \setminus S$. $\delta_i(S)$ is also referred to as the cut set of S corresponding to the i^{th} vehicle. Equation (2) states that for any $S \subseteq T$ at least two edges must be chosen from $\delta_1(S)$ if there is at least one vertex in S that is not required to be connected to the second depot, *i.e.*, $\sum_{e \in \delta_1(S)} x_e \geq 2$ if $\sum_{T \supseteq U \supseteq S} z_U = 0$. Similarly, equation (3) states that for any $S \subseteq T$, at least two edges must be chosen from $\delta_2(S)$ if all the vertices in S are required to be visited by the second vehicle. Equations (4,5) state that the degree of each of the targets must be equal to 2. In the above formulation, we have not included the degree constraints on the depots for the following reason: as the costs satisfy the triangle inequality, one can always shortcut the edges incident on the depots in any solution to ensure that the degree of a used depot is 2. Now, consider a Linear Programming (LP) relaxation of the problem where the constraints in equations (4,5,6,7) are relaxed as follows:

$$C_{lp} = \min \sum_{e \in E_1} cost_e^1 x_e + \sum_{e \in E_2} cost_e^2 y_e \quad (8)$$

$$\begin{aligned}
\sum_{e \in \delta_1(S)} x_e + 2 \sum_{T \supseteq U \supseteq S} z_U &\geq 2 & \forall S \subseteq T, \\
\sum_{e \in \delta_2(S)} y_e &\geq 2 \sum_{T \supseteq U \supseteq S} z_U & \forall S \subseteq T, \\
x_e &\geq 0 \quad \forall e \in E_1, \quad y_e \geq 0 \quad \forall e \in E_2, \\
z_U &\geq 0 \quad \forall U \subseteq T.
\end{aligned}$$

A dual of the above LP relaxation can be formulated as follows:

$$C_{dual} = \max 2 \sum_{S \subseteq T} Y_1(S) \quad (9)$$

$$\sum_{S: e \in \delta_1(S)} Y_1(S) \leq cost_e^1 \quad \forall e \in E_1, \quad (10)$$

$$\sum_{S: e \in \delta_2(S)} Y_2(S) \leq cost_e^2 \quad \forall e \in E_2, \quad (11)$$

$$\sum_{S \subseteq U} Y_1(S) \leq \sum_{S \subseteq U} Y_2(S) \quad \forall U \subseteq T, \quad (12)$$

$$Y_1(S), Y_2(S) \geq 0 \quad \forall S \subseteq T. \quad (13)$$

We use the above dual problem to find a Heterogeneous Spanning Forest (HSF). A HSF is a collection of two trees where the first tree spans a subset of targets and d_1 using edges only from E_1 , and the second tree connects the remaining set of targets to d_2 using edges only from E_2 . In the next section, we present a primal-dual algorithm that finds a HSF. We later show that the cost of this HSF is at most equal to the optimal cost of the above dual. This leads to a 2-approximation algorithm for the 2DHTSP.

III. A PRIMAL-DUAL ALGORITHM FOR FINDING A HETEROGENEOUS SPANNING FOREST

The initialization, the main steps and the final pruning step of the primal-dual algorithm are presented in **Algorithms 1, 2 and 3**. In the following discussion, we first aim to highlight the important features of the algorithm and give an outline of the important steps. The proofs and the correctness of the algorithm will be discussed later. At any iteration, the algorithm maintains two sets of connected components corresponding to the two vehicles. In the algorithm, \mathcal{C}_1 denotes the set of connected components corresponding to the first vehicle. Similarly, \mathcal{C}_2 represents the set of connected components corresponding to the second vehicle. For $i = 1, 2$, we use F_i to denote the set of edges connecting the vertices in \mathcal{C}_i . Initially, both \mathcal{C}_1 and \mathcal{C}_2 consist of components where each vertex is in its own connected component, *i.e.*, $\mathcal{C}_1 = \{\{v\} : v \in V_1\}$ and $\mathcal{C}_2 = \{\{v\} : v \in V_2\}$. That is, both F_1 and F_2 are empty. All the components in \mathcal{C}_1 and \mathcal{C}_2 are initially active except the components that contain the depots. Also, each vertex in V_1 is initially unmarked. At the beginning of each iteration, for any $\forall C \in \mathcal{C}_1$, the internal variable $w(C)$ keeps track of $\sum_{S \subseteq C} Y_1(S)$, *i.e.*, $w(C) = \sum_{S \subseteq C} Y_1(S)$. Similarly, $\forall C \in \mathcal{C}_1$, $Bound(C)$ keeps track of $\sum_{S \subseteq C} Y_2(S)$. Essentially, variables $w(C)$ and $Bound(C)$ are used to enforce the constraints in (12). At the start of the first iteration, both $w(C)$ and $Bound(C)$ are set to zero, *i.e.*, $w(C) := 0$ and $Bound(C) := 0 \quad \forall C \in \mathcal{C}_1$ (Refer to the initialization steps in algorithm 1).

As the primal-dual algorithm follows a greedy procedure, and since the cost of traveling between any two targets is cheaper for the first vehicle (*i.e.*, $cost_{\{i,j\}1}^1 \leq cost_{\{i,j\}2}^2 \forall i, j \in T$), the algorithm prefers to connect any two targets using edges from E_1 before using any edges from E_2 . Therefore, as the algorithm progresses, the components in \mathcal{C}_1 tend to merge and grow bigger than their corresponding components in \mathcal{C}_2 . In this article, we refer to the components in \mathcal{C}_1 as parents and the components in \mathcal{C}_2 as their children. For components $C_1 \in \mathcal{C}_1$ and $C_2 \in \mathcal{C}_2$, we define C_1 as the parent of C_2 and C_2 as a child of C_1 if $C_2 \subseteq C_1$ and $d_2 \notin C_2$. For any component $C_1 \in \mathcal{C}_1$, we use $Children(C_1)$ to denote all the children of C_1 present in \mathcal{C}_2 . For any component $C_2 \in \mathcal{C}_2, d_2 \notin C_2$, we use $Parent(C_2)$ to denote the parent of C_2 present in \mathcal{C}_1 . According to the definition, if C_2 contains the depot d_2 , C_2 doesn't have a parent; however, to simplify the presentation, we let $Parent(C_2)$ be an empty set if C_2 contains d_2 . At the start of the algorithm, for any target $v \in T$, $Children(\{v\})$ is assigned to be equal to $\{v\}$ and $Parent(\{v\})$ is assigned to be equal to $\{v\}$. Also, the components that consist of just the depots neither have a parent or a child (*Refer to the initialization steps in algorithm 1*).

In each iteration of the algorithm, the dual variable corresponding to each of the *active* components in \mathcal{C}_1 and \mathcal{C}_2 are increased as much as possible by the *same* amount until one of the constraints stated in (10-12) become tight (*Refer to lines 2-5 of the algorithm 2*). If a constraint in (10) becomes tight for some edge $e \in E_1$, F_1 is augmented with this new edge and the two components (say C_{1x}, C_{1y} in \mathcal{C}_1) connected by e are merged to form a single connected component. The children of each of the two components C_{1x}, C_{1y} now together become the children of the resulting component $C_{1x} \cup C_{1y}$. The resulting component becomes inactive if it contains the depot d_1 ; otherwise, it is active. In the case when the resulting component becomes inactive, all the children of the resulting component also become inactive (*Refer to lines 17-27 of the algorithm 2*).

Algorithm 1 *Primal-dual algorithm: Initialization*

```

 $F_1 \leftarrow \emptyset; F_2 \leftarrow \emptyset$ 
 $\mathcal{C}_1 \leftarrow \{\{v\} : v \in V_1\}; \mathcal{C}_2 \leftarrow \{\{v\} : v \in V_2\}$ 
for  $v \in V_1$  do
  Unmark  $v$ 
   $p(v) \leftarrow 0$ 
   $w(\{v\}) \leftarrow 0$ 
   $Bound(\{v\}) \leftarrow 0$ 
  If  $v = d_1$ , then  $Children(\{v\}) \leftarrow \emptyset$ , else  $Children(\{v\}) \leftarrow \{v\}$ 
  If  $v = d_1$ , then  $active_1(\{v\}) = 0$ , else  $active_1(\{v\}) = 1$ 
end for
for  $v \in V_2$  do
   $q(v) \leftarrow 0$ 
  If  $v = d_2$ , then  $Parent(\{v\}) \leftarrow \emptyset$ , else  $Parent(\{v\}) \leftarrow \{v\}$ 
  If  $v = d_2$ , then  $active_2(\{v\}) = 0$ , else  $active_2(\{v\}) = 1$ 
end for

```

Similarly, if one of constraints in (11) becomes tight for some edge $e \in E_2$, F_2 is augmented with this new edge and the two components (say C_{2x}, C_{2y} in \mathcal{C}_2) connected by e are merged to form a single connected component (*Refer to lines 29-42 of the algorithm 2*). The resulting component becomes inactive if it contains the depot d_2 ; otherwise, it is active. In the case when the resulting component is active, the parent of either C_{2x} or C_{2y} is assigned as the parent of the resulting component (It turns out that due to our assumptions on the costs, when the algorithm enters this part of the implementation, both C_{2x} and C_{2y} must be active and must be the children of the same parent; we will show this result later in the article). In the case when the resulting component becomes inactive, and say C_{2x} was the active

Algorithm 2 : Primal-dual algorithm - Main steps

```

1: while  $\exists C \in \mathcal{C}_1$  such that  $active_1(C) = 1$  do
2: Find edge  $e_1 = \{i, j\}_1 \in E_1$  with  $i \in C_{1x}, j \in C_{1y}$  where  $C_{1x}, C_{1y} \in \mathcal{C}_1, C_{1x} \neq C_{1y}$  that minimizes
    $\varepsilon_1 = \frac{(cost_{e_1}^1 - p(i) - p(j))}{active_1(C_{1x}) + active_1(C_{1y})}$ 
3: Find edge  $e_2 = \{i, j\}_2 \in E_2$  with  $i \in C_{2x}, j \in C_{2y}$  where  $C_{2x}, C_{2y} \in \mathcal{C}_2, C_{2x} \neq C_{2y}$  that minimizes
    $\varepsilon_2 = \frac{(cost_{e_2}^2 - q(i) - q(j))}{active_2(C_{2x}) + active_2(C_{2y})}$ 
4: Find a maximal  $\mathfrak{C} \subseteq \mathcal{C}_1$  such that for all  $\overline{C} \in \mathfrak{C}$  the following properties hold:  $active_1(\overline{C}) = 1$  and  $Children(\overline{C})$  is
   an empty set. If  $\mathfrak{C}$  is empty, then  $\varepsilon_3 \leftarrow \infty$ , else find  $\overline{C} \in \mathfrak{C}$  that minimizes  $\varepsilon_3 = Bound(\overline{C}) - w(\overline{C})$ 
5:  $\varepsilon_{min} = \min(\varepsilon_1, \varepsilon_2, \varepsilon_3)$ 
6: for  $C \in \mathcal{C}_1$  do
7:  $w(C) \leftarrow w(C) + \varepsilon_{min} \times active_1(C)$ 
8: For all  $v \in C$ ,  $p(v) \leftarrow p(v) + \varepsilon_{min} \times active_1(C)$ 
9:  $Bound(C) \leftarrow Bound(C) + \sum_{\hat{C} \in Children(C)} \varepsilon_{min} \times active_2(\hat{C})$ 
10: end for
11: for  $C \in \mathcal{C}_2$  do
12: For all  $v \in C$ ,  $q(v) \leftarrow q(v) + \varepsilon_{min} \times active_2(C)$ 
13: end for
14: switch  $\varepsilon_{min}$ 
15: //Comment: If more than one value in  $\{\varepsilon_1, \varepsilon_2, \varepsilon_3\}$  is equal to  $\varepsilon_{min}$ , then give priority first to Case  $\varepsilon_1$ ,
   then to Case  $\varepsilon_2$  and finally to Case  $\varepsilon_3$ 
16: Case  $\varepsilon_1$ :
17:  $F_1 \leftarrow F_1 \cup \{e_1\}$ 
18:  $\mathcal{C}_1 \leftarrow \mathcal{C}_1 \cup \{C_{1x} \cup C_{1y}\} - C_{1x} - C_{1y}$ 
19:  $w(C_{1x} \cup C_{1y}) \leftarrow w(C_{1x}) + w(C_{1y})$ 
20:  $Children(C_{1x} \cup C_{1y}) \leftarrow Children(C_{1x}) \cup Children(C_{1y})$ 
21: For all  $C \in Children(C_{1x} \cup C_{1y})$ ,  $Parent(C) \leftarrow C_{1x} \cup C_{1y}$ 
22:  $Bound(C_{1x} \cup C_{1y}) \leftarrow Bound(C_{1x}) + Bound(C_{1y})$ 
23: if  $d_1 \in C_{1x} \cup C_{1y}$ , then
24:  $active_1(C_{1x} \cup C_{1y}) = 0$ 
25:  $active_2(C) = 0$  for all  $C \in Children(C_{1x} \cup C_{1y})$ 
26: else  $active_1(C_{1x} \cup C_{1y}) = 1$ 
27: end
28: Case  $\varepsilon_2$ :
29:  $F_2 \leftarrow F_2 \cup \{e_2\}$ 
30:  $\mathcal{C}_2 \leftarrow \mathcal{C}_2 \cup \{C_{2x} \cup C_{2y}\} - C_{2x} - C_{2y}$ 
31: if  $d_2 \in C_{2x} \cup C_{2y}$  then
32:  $active_2(C_{2x} \cup C_{2y}) \leftarrow 0$ 
33:  $Parent(C_{2x} \cup C_{2y}) \leftarrow \emptyset$ 
34: Let  $C \in \{C_{2x}, C_{2y}\}$  such that  $d_2 \notin C$ ;  $Children(Parent(C)) \leftarrow Children(Parent(C)) - C$ 
35: else if  $active_2(C_{2x}) = 1$  and  $active_2(C_{2y}) = 1$ 
36:  $active_2(C_{2x} \cup C_{2y}) \leftarrow 1$ 
37:  $C_{temp} \leftarrow Parent(C_{2x})$ 
38: //Comment:  $Parent(C_{2x})$  must be the same as  $Parent(C_{2y})$ 
39:  $Parent(C_{2x} \cup C_{2y}) \leftarrow C_{temp}$ 
40:  $Children(C_{temp}) \leftarrow Children(C_{temp}) \cup \{C_{2x} \cup C_{2y}\} - C_{2x} - C_{2y}$ 
41: end if
42: end if
43: Case  $\varepsilon_3$ :
44:  $active_1(\overline{C}) \leftarrow 0$ 
45: Mark all the unlabeled vertices of  $\overline{C}$  with label  $\overline{C}$ 
46: end switch
47: end while

```

Algorithm 3 : Primal-dual algorithm - Pruning step

- 1: F'_1 is obtained from F_1 by removing as many edges as possible from F_1 so that the following properties hold: 1) All the unmarked vertices of V_1 are connected to the first depot d_1 ; 2) If any vertex with label C is connected to the depot d_1 , then any other vertex with a label $C' \supseteq C$ is also connected to the depot d_1 .
 - 2: F'_2 is obtained from F_2 by removing as many edges as possible from F_2 such that any target not spanned by F'_1 is spanned by F'_2 .
-

component during the iteration which *did not* contain the depot, the parent of C_{2x} loses C_{2x} as its child.

If one of the third set of constraints in (12) become tight for some component \overline{C} in \mathcal{C}_1 , then the algorithm deactivates \overline{C} and marks each of the unmarked vertices in the component with \overline{C} (Refer to lines 44-45 of the algorithm 2). The algorithm terminates when all the components in \mathcal{C}_1 become inactive. After termination, the algorithm makes one final pass at all the edges (refer to algorithm 3) and removes any edge that is not required to be in the HSF. Basically, during the final step of the primal dual algorithm, any unnecessary edges in F_1 and F_2 are pruned further to find a tree for each of the vehicles. Specifically, the tree corresponding the the first vehicle is obtained from F_1 by removing as many edges as possible from F_1 so that the following properties hold: 1) All the unmarked vertices of V_1 are connected to the first depot d_1 ; 2) If any vertex with label C is connected to the depot d_1 , then any other vertex with a label $C' \supseteq C$ is also connected to the depot d_1 . The tree corresponding the second vehicle is obtained from F_2 by removing as many edges as possible from F_2 such that any target not spanned by F'_1 is connected to d_2 in F'_2 .

Since the sum of the number of components in \mathcal{C}_1 , the number of active components in \mathcal{C}_1 and the number of components in \mathcal{C}_2 decrease at least by one during each iteration, the primal-dual algorithm must terminate after at most $3|T|+2$ iterations. Using the techniques given in [10], this primal-dual algorithm can be implemented in $|T|^2 \log |T|$ steps.

Remark: Case ε_2 is never chosen by the algorithm if one of the components in C_{2x}, C_{2y} is inactive and at the same time, the depot d_2 is not present in either C_{2x} or C_{2y} . For this reason, the *if* loop in lines 31-42 does not consider this case at all. The proof of this statement is shown in lemma 1.

Remark: In lines 36-40 of the main algorithm, an active component $C_{2x} \in \mathcal{C}_2$ merges with another active component $C_{2y} \in \mathcal{C}_2$. We claim that the algorithm will enter these lines only if both C_{2x} and C_{2y} have the same parent. We show this in lemma 1.

Remark: At the termination of the algorithm, F'_1 will connect a subset of targets using edges from E_1 to the depot d_1 . F'_2 will connect each of the targets not spanned by F'_1 to depot d_2 using edges from E_2 . The proof of this statement is shown in lemma 2.

In the following subsection, we first discuss some properties of this primal-dual algorithm before proving its approximation ratio.

A. Properties of the primal-dual algorithm

Observation 1: At the start of an iteration of the algorithm, for any two disjoint components $C_{1x}, C_{1y} \in \mathcal{C}_1$, consider the constraint in (10) corresponding to the edge $e = \{u, v\}_1$ that could potentially connect vertex u in C_{1x} to vertex v in C_{1y} : $\sum_{S: e \in \delta_1(S)} Y_1(S) \leq \text{cost}_e^1$. Since e has not yet been added to F_1 , this constraint can be re-written as $\sum_{S: u \in S} Y_1(S) + \sum_{S: v \in S} Y_1(S) \leq \text{cost}_e^1$. To simplify this constraint further, let us define $p(v) = \sum_{S: v \in S} Y_1(S) \forall v \in V_1$. Now, the constraint can be formulated as $p(u) + p(v) \leq \text{cost}_e^1$.

We can interpret $p(v)$ as the total price vertex v is willing to pay to buy any edge that is incident on vertex v . Therefore, to add an edge $\{u, v\}_1$ during the iteration, both the vertices u and v must increase the sum of the prizes they are willing to pay by $\text{cost}_e^1 - p(u) - p(v)$. Hence, each of the dual variables of the active components have to be increased by an amount given by $\frac{\text{cost}_e^1 - p(u) - p(v)}{\text{active}_{e_1}(C_{1x}) + \text{active}_{e_1}(C_{1y})}$ in order to make the constraint, $p(u) + p(v) \leq \text{cost}_e^1$, tight. Hence, in step 2 of the algorithm 2, we basically find the minimum amount by which each of the dual variables of the active components in \mathcal{C}_1 have to be increased so that none of the constraints are violated and at least one of the constraints in (10) just become tight. Similarly, in step 3 of the algorithm 2, we find the minimum amount by which each of the dual variables of the active components in \mathcal{C}_2 have to be increased so that none of the constraints are violated and at least one of the constraints in (11) just become tight.

Observation 2: For any active component $C_1 \in \mathcal{C}_1$, as long as C_1 has at least one child, the constraint associated with C_1 in (12) never gets tight. In the algorithm, the variable $\text{Bound}(C_1)$ keeps track of the amount by which $\sum_{S \subseteq C_1} Y_2(S)$ is increased during each iteration. Once an active component C_1 loses all its children, $\text{Bound}(C_1)$ specifies the maximum value that can be attained by $w(C_1) = \sum_{S \subseteq C_1} Y_1(S)$.

Observation 3: An active component in \mathcal{C}_1 can get deactivated during an iteration due to one of the following reasons: 1) the component merges with another component in \mathcal{C}_1 that consists of the depot d_1 , 2) the dual variable of the active component cannot be increased more than a certain value as it would violate its associated constraint in (12). On the other hand, an active component in \mathcal{C}_2 can get deactivated during an iteration due to one of the following reasons: 1) the component merges with another component in \mathcal{C}_2 that consists of the depot d_2 , 2) the parent of the component gets deactivated, *i.e.*, the parent of the component gets connected to depot d_1 .

Observation 4: If an active component $C_1 \in \mathcal{C}_1$ merges with another component containing d_1 , then the merged component and all the children of the merged component (which includes the children of C_1) also get deactivated. For this reason, each of the children of any parent containing d_1 is always inactive during any iteration of the algorithm. However, if a component $C_2 \in \mathcal{C}_2$ merges with another component containing d_2 and gets deactivated, then the parent of C_2 loses C_2 as its child but can still be active. Moreover, the dual variable corresponding to parent of C_2 can continue to increase until one of the constraints in (10) or (12) becomes tight.

Observation 5: In each iteration, for any target $v \in T$, $p(v)$ is incremented by e_{\min} if v belongs to a component in \mathcal{C}_1 that is active; else $p(v)$ does not change. Similarly, for any target $v \in T$, $q(v)$ is incremented by e_{\min} if v belongs to a component in \mathcal{C}_2 that is active; else, $q(v)$ does not change.

Lemma 1: Consider any target $u \in T$. At the start of the k^{th} iteration, let $C_1^k(u)$ denote the component in \mathcal{C}_1 containing u , and $C_2^k(u)$ represent the component in \mathcal{C}_2 containing u . Then, each of the following statements are true:

1. Consider a case when $C_1^k(u)$ is a parent of $C_2^k(u)$. If $C_2^k(u)$ is inactive because its parent $C_1^k(u)$ contains d_1 , then $C_2^k(u)$ can never merge with any other component during the k^{th} iteration.
2. For any two distinct targets u and v , if components $C_2^k(u) \in \mathcal{C}_2$ and $C_2^k(v) \in \mathcal{C}_2$ are active, then $C_2^k(u)$ can merge with $C_2^k(v)$ during the k^{th} iteration using an edge $\{u, v\}_2$, only if both $C_2^k(u)$ and $C_2^k(v)$ have the same parent.
3. Consider a case when $C_2^k(u)$ is active and is a child to an active $C_1^k(u)$. Then, at the end of the k^{th} iteration (or at the start of the $k+1^{th}$ iteration), the parent $C_1^{k+1}(u)$ does not have any child which contains target u only if $C_2^k(u)$ merges with another component in \mathcal{C}_2 that consists of the depot d_2 .
4. If $active_1(C_1^k(u)) \geq active_2(C_2^k(u))$, then at the end of the k^{th} iteration (or, at the start of the next iteration), $active_1(C_1^{k+1}(u)) \geq active_2(C_2^{k+1}(u))$.

Proof: Let us prove this lemma by induction. During the first iteration, there is no component in \mathcal{C}_2 that has a parent consisting of d_1 . Therefore, lemma 1.1 is true by default. At the start of first iteration, each of the active components in both \mathcal{C}_1 and \mathcal{C}_2 contain only one target. Also, for any $u \in T$, the component $\{u\} \in \mathcal{C}_1$ is the parent of $\{u\} \in \mathcal{C}_2$. Since the cost of the edge joining any two targets for the first vehicle is at most equal to the cost of the edge joining the same targets for the second vehicle, the algorithm will always prefer to edge $\{u, v\}_1$ as compared to $\{u, v\}_2$. Therefore, no edge corresponding to the second vehicle joining two targets will be added during the first iteration of the algorithm. Hence, lemma 1.2 is true by default for $k = 1$. The only way the parent $\{u\} \in \mathcal{C}_1$ loses $\{u\} \in \mathcal{C}_2$ as the child is if $\{u\} \in \mathcal{C}_2$ merges with the component $\{d_2\}$ during the first iteration. In this case, a new inactive component $\{u, d_2\} \in \mathcal{C}_2$ is formed and $\{u\} \in \mathcal{C}_1$ loses $\{u\} \in \mathcal{C}_2$ as the child. Hence lemma 1.3 is correct for $k = 1$. At the start of the first iteration, each of the components which contain one target is active. Therefore, $active_1(C_1^1(u)) \geq active_2(C_2^1(u))$. Also, during the first iteration, one of the following cases can happen: (a) an edge is added to F_1 joining two targets or a target and d_1 , (b) an edge is added to F_2 joining a target and d_2 . In either case, one can verify that the components are merged such that $active_1(C_1^2(u)) \geq active_2(C_2^2(u))$ for all $u \in T$. Hence lemma 1.4 is correct for $k = 1$.

Now, let us assume that all the statements in lemma 1.1-1.4 are true for the l^{th} iteration for any $l = 1, \dots, k-1$. From lemma 1.4, this assumption implies that $active_1(C_1^l(u)) \geq active_2(C_2^l(u))$ for any $l = 1, \dots, k$. We then show that all these statements are also true for the k^{th} iteration.

Proof of lemma 1.1: In this case $C_1^k(u)$ is a parent of $C_2^k(u)$, and $C_2^k(u)$ is inactive because its parent $C_1^k(u)$ contains d_1 . Since $C_1^k(u)$ contains d_1 , from observation 4, it follows that all the children of $C_1^k(u)$ are inactive. Therefore, if $C_2^k(u)$ which is inactive has to merge with some other component $C_2^k(v)$ corresponding to target v , then $Parent(C_2^k(u)) \neq Parent(C_2^k(v))$ and $C_2^k(v)$ must be active. Since $active_1(C_1^l(u)) \geq active_2(C_2^l(u))$ for any $l = 1, \dots, k$, from observation 5, it follows that $p(u) \geq q(u)$ at the start of the k^{th} iteration. Similarly, for target v , we must also have $p(v) \geq q(v)$ at the start of the k^{th} iteration. Since, $cost_{\{u,v\}_1}^1 \leq cost_{\{u,v\}_2}^2$, we now have

$$\begin{aligned} \varepsilon_1 &= \frac{cost_{\{u,v\}_1}^1 - p(u) - p(v)}{active_1(C_1^k(u)) + active_1(C_1^k(v))} \\ &\leq \frac{cost_{\{u,v\}_2}^2 - q(u) - q(v)}{active_2(C_2^k(u)) + active_2(C_2^k(v))} = \varepsilon_2. \end{aligned} \tag{14}$$

As a result, the algorithm will prefer to add edge $\{u, v\}_1$ as compared to adding $\{u, v\}_2$. But, once $\{u, v\}_1$ is added,

the component $C_2^k(v)$ will become a child of $C_1^k(u) \cup C_1^k(v)$ and as a result will be deactivated. Therefore, $C_2^k(u)$ will remain inactive and will never merge with any other component during the k^{th} iteration.

Proof of lemma 1.2: For each target $u \in T$, if $active_1(C_1^l(u)) \geq active_2(C_2^l(u))$ for any $l = 1 \dots k$, then from observation 5, it follows that $p(u) \geq q(u)$ at the start of the k^{th} iteration. Let $C_2^k(u) \in \mathcal{C}_2$ and $C_2^k(v) \in \mathcal{C}_2$ be active and disjoint components, and let $Parent(C_2^k(u)) \neq Parent(C_2^k(v))$. As $cost_{\{u,v\}_1}^1 \leq cost_{\{u,v\}_2}^2$ for any $u, v \in T$, again using equation 14, we can conclude that $\varepsilon_1 \leq \varepsilon_2$. If $\varepsilon_1 < \varepsilon_2$, edge $\{u, v\}_2$ cannot be added during the iteration. If $\varepsilon_1 = \varepsilon_2$, as mentioned in line 15 of algorithm 2, preference is given to adding edge $\{u, v\}_1$ as opposed to adding edge $\{u, v\}_2$. Therefore, in either case, edge $\{u, v\}_2$ cannot be added during the iteration. Hence, two active components in \mathcal{C}_2 can merge only if they have the same parent.

Proof of lemma 1.3: Due to the manner in which the components are deactivated in the algorithm (Refer to line 34 in algorithm 2 and observation 4), if $C_2^k(u)$ is active and is a child to $C_1^k(u)$, at the end of the k^{th} iteration, the parent $C_1^{k+1}(u)$ does not have any child which contains target u only if $C_2^k(u)$ merges with another component in \mathcal{C}_2 that consists of the depot d_2 .

Proof of lemma 1.4:

If $C_2^k(u)$ is **inactive**, from observation 3, either $C_2^k(u)$ must contain the depot d_2 or its parent $C_1^k(u)$ must contain the depot d_1 .

- If $C_2^k(u)$ already contains d_2 , then $C_2^{k+1}(u)$ must also be inactive. Therefore, $active_1(C_1^{k+1}(u)) \geq active_2(C_2^{k+1}(u)) = 0$.
- If $C_2^k(u)$ is inactive because its parent $C_1^k(u)$ contains d_1 , then we have already shown that $C_2^k(u)$ can never merge with any other component during the k^{th} iteration. Therefore, $active_1(C_1^{k+1}(u)) \geq active_2(C_2^{k+1}(u))$.

If $C_2^k(u)$ is **active**, then $active_1(C_1^k(u)) \geq active_2(C_2^k(u))$ implies that $C_1^k(u)$ is also active. In addition, since the lemma is true for any $l = 1, \dots, k-1$, from lemma 1.3 it follows that $C_1^k(u)$ is a parent of $C_2^k(u)$. Since the component, $C_1^k(u)$, has at least one active child in $C_2^k(u)$, $C_1^k(u)$ can never become inactive due to its associated constraint in (12) during the k^{th} iteration. The only way $C_1^k(u)$ can lead to an inactive $C_1^{k+1}(u)$ is if $C_1^k(u)$ merges with another component containing d_1 during the iteration in which case all the children of $C_1^k(u)$ including $C_2^k(u)$ also get deactivated. Therefore, $active_1(C_1^{k+1}(u)) \geq active_2(C_2^{k+1}(u))$. ■

Corollary 1: An edge joining two targets $u, v \in T$ can be added to F_2 only if both u and v are already connected in F_1 , i.e., both u and v belong to the same component in \mathcal{C}_1 .

Proof: We use the same notation we used in lemma 1. If target u is not connected to v at the start of the k^{th} iteration in both (V_1, F_1) and (V_2, F_2) , then using the results from lemma 1 and equation 14, $\varepsilon_1 \leq \varepsilon_2$. Therefore, the algorithm always prefers to connect targets u, v first using $\{u, v\}_1$ as compared to using $\{u, v\}_2$. Hence proved. ■

Lemma 2: The algorithm produces a feasible, heterogeneous spanning forest, i.e., the trees specified by the collection of edges in F'_1 and F'_2 connect each of the targets to one of the depots. Any vertex spanned by the edges in F'_1 is not spanned by the edges in F'_2 and vice versa.

Proof: The algorithm terminates when all the sets of \mathcal{C}_1 become inactive. This is only possible if each of the targets in T is either connected to d_1 or d_2 . Note that F'_1 is formed from F_1 such that each of the unmarked vertices remain

connected to d_1 . The only vertices not spanned by F'_1 are some of the marked vertices. These vertices were marked because the components in \mathcal{C}_1 that span these vertices were deactivated for making their associated constraints in (12) tight. In addition, a component in \mathcal{C}_1 can become deactivated due to a constraint in (12) only if it has already lost all its children, *i.e.*, each of these vertices in the component is already connected to d_2 . Therefore, by the construction of F'_2 , each of the marked vertices not spanned by F'_1 must be connected to d_2 and spanned by F'_2 . Hence, the algorithm produces a feasible, heterogeneous spanning forest.

Also, because of the way in which we prune the edges in the final step of the algorithm, if a vertex with label C is not spanned by F'_1 , then any other vertex with label $C' \subseteq C$ is also not spanned by F'_1 . Let \mathfrak{X} denote the set of vertices not spanned by F'_1 . Based on the label of each vertex in \mathfrak{X} , \mathfrak{X} can be partitioned into disjoint, deactivated components $\overline{C}_1, \overline{C}_2, \dots, \overline{C}_m$ where each \overline{C}_i denotes the maximal label of its respective component.

Consider any deactivated component $\overline{C}_i \subseteq \mathfrak{X}$. \overline{C}_i can get deactivated during an iteration only if \overline{C}_i does not have children and $\sum_{S \subseteq \overline{C}_i} Y_1(S) = w(\overline{C}_i) = \text{Bound}(\overline{C}_i) = \sum_{S \subseteq \overline{C}_i} Y_2(S)$. Note that \overline{C}_i could have lost all its children only if all the targets in \overline{C}_i are already connected to d_2 in F_2 . Also, during the iteration when \overline{C}_i gets deactivated, no target $u \in \overline{C}_i$ is connected to any other target $v \in T \setminus \overline{C}_i$ in F_1 . From corollary 1, it follows that for any $u \in \overline{C}_i$, any adjacent vertex of u in the deactivated component of \mathcal{C}_2 can either be d_2 or another vertex $v \in \overline{C}_i$. Therefore, during the construction of F'_2 all the edges that are incident on any vertex already spanned by F'_1 can be dropped. Hence, any vertex spanned by the edges in F'_1 is not spanned by the edges in F'_2 and vice versa. ■

The main result of this article is in the following theorem.

Theorem III.1: The primal-dual algorithm produces a tree with edges denoted by F'_1 for the first vehicle and a tree with edges denoted by F'_2 for the second vehicle such that the cost of the edges in these trees is bounded by the cost for the dual problem, *i.e.*,

$$\sum_{e \in F'_1} \text{cost}_e^1 + \sum_{e \in F'_2} \text{cost}_e^2 \leq 2 \sum_{S \subseteq T} Y_1(S).$$

Since $2 \sum_{S \subseteq T} Y_1(S)$ is a lower bound to the optimal cost of the 2DHTSP, it follows that the cost of the HSF found by the primal dual algorithm is at most equal to the optimal cost of the 2DHTSP.

B. Proof of the Approximation Ratio

As mentioned in lemma 2, let \mathfrak{X} denote the set of vertices not spanned by F'_1 . Based on the label of each vertex in \mathfrak{X} , \mathfrak{X} can be partitioned into disjoint, deactivated components $\overline{C}_1, \overline{C}_2, \dots, \overline{C}_m$ where each \overline{C}_i denotes the maximal label of its respective component. In order to prove the above theorem, we first simplify the dual cost obtained by the algorithm as follows:

$$\begin{aligned} 2 \sum_{S \subseteq T} Y_1(S) &= 2 \sum_{S \subseteq T, S \not\subseteq \overline{C}_i, i=1, \dots, m} Y_1(S) + 2 \sum_{i=1}^m \sum_{S \subseteq \overline{C}_i} Y_1(S) \\ &= 2 \sum_{S \subseteq T, S \not\subseteq \overline{C}_i, i=1, \dots, m} Y_1(S) + 2 \sum_{i=1}^m \sum_{S \subseteq \overline{C}_i} Y_2(S). \end{aligned} \quad (15)$$

Now, we express the cost of the edges in the first tree in terms of the dual variables as follows. Note that edge e is added to F_1 and consequently appears in F'_1 only if the corresponding constraint in (10) is tight, *i.e.*, $\text{cost}_e^1 = \sum_{S: e \in \delta_1(S)} Y_1(S)$.

Therefore,

$$\begin{aligned}\sum_{e \in F'_1} cost_e^1 &= \sum_{e \in F'_1} \sum_{S: e \in \delta_1(S)} Y_1(S) \\ &= \sum_{S \subseteq T} Y_1(S) |F'_1 \cap \delta_1(S)|.\end{aligned}$$

Since $F'_1 \cap \delta_1(S) = 0$ for any $S \subseteq \overline{C}_i$, we can further simplify the above equation to

$$\sum_{e \in F'_1} cost_e^1 = \sum_{S \subseteq T, S \not\subseteq \overline{C}_i, i=1, \dots, m} Y_1(S) |F'_1 \cap \delta_1(S)|. \quad (16)$$

Similarly, we can also express the cost of the edges in the second tree in terms of the dual variables as follows. From lemma 2, note that F'_2 can be decomposed into a set of disjoint sets F'_{2i} where each F'_{2i} consists of edges that form a tree spanning each target from \overline{C}_i and the depot d_2 . An edge e is added to F_2 and consequently appears in F'_{2i} only if the corresponding constraint in (11) is tight, $cost_e^2 = \sum_{S: e \in \overline{\delta}_{2i}(S), S \subseteq \overline{C}_i} Y_2(S)$ where $\overline{\delta}_{2i}(S)$ consists of all the edges with one endpoint in S and another end point in $\overline{C}_i \cup \{d_2\} \setminus S$.

$$\begin{aligned}\sum_{e \in F'_2} cost_e^2 &= \sum_{i=1}^m \sum_{e \in F'_{2i}} cost_e^2 \\ &= \sum_{i=1}^m \sum_{e \in F'_{2i}} \sum_{S: e \in \overline{\delta}_{2i}(S), S \subseteq \overline{C}_i} Y_2(S) \\ &= \sum_{i=1}^m \sum_{S \subseteq \overline{C}_i} Y_2(S) |F'_{2i} \cap \overline{\delta}_{2i}(S)|.\end{aligned} \quad (17)$$

Therefore, from equations (15, 16, 17), the proof for the theorem reduces to showing the following result:

$$\sum_{S \subseteq T, S \not\subseteq \overline{C}_i, i=1, \dots, m} Y_1(S) |F'_1 \cap \delta_1(S)| + \sum_{i=1}^m \sum_{S \subseteq \overline{C}_i} Y_2(S) |F'_{2i} \cap \overline{\delta}_{2i}(S)| \quad (18)$$

$$\leq 2 \sum_{S \subseteq T, S \not\subseteq \overline{C}_i, i=1, \dots, m} Y_1(S) + 2 \sum_{i=1}^m \sum_{S \subseteq \overline{C}_i} Y_2(S). \quad (19)$$

The above result can be shown by proving that during any iteration, the increase in the primal cost (the left-hand side of the above inequality) is at most equal to the increase in the dual cost (the right-hand side of the above inequality). To see this, let us choose any iteration of the primal-dual algorithm. At the start of this iteration, let N_a be the set of all the active components in \mathcal{C}_1 such that each active component in this set is not a subset of \mathfrak{X} and N_d be the set of all the inactive components in \mathcal{C}_1 such that each inactive component in this set is not a subset of \mathfrak{X} . Note that one of inactive components of N_d must consist of the depot d_1 . For $i = 1, \dots, m$, let M_{ai} denote the set of all the active components in \mathcal{C}_2 such that each active component in this set is a subset of \overline{C}_i . Also, let M_d denote the inactive component in \mathcal{C}_2 that consists of the depot d_2 .

Now, form a graph H_1 with components in $N_a \cup N_d$ as its vertices and edges $e \in F'_1 \cap \delta_1(C)$ for $C \in N_a \cup N_d$ as edges of H_1 . Similarly, form a graph H_{2i} with components in $M_{ai} \cup M_d$ as its vertices and edges $e \in F'_{2i} \cap \delta_2(C)$ for

$C \in M_{ai} \cup M_d$ as edges of H_{2i} . Let $\deg(v, G)$ represent the degree of vertex v in graph G . During the iteration, the dual variable corresponding to each of the active components is increased by ε_{min} . As the result, the left hand side of the inequality will increase by $\varepsilon_{min}(\sum_{v \in N_a} \deg(v, H_1) + \sum_{i=1}^m \sum_{v \in M_{ai}} \deg(v, H_{2i}))$ whereas the right hand side of the inequality will increase by $2\varepsilon_{min}(N_a + \sum_{i=1}^m M_{ai})$. Therefore, basically, the proof is complete if we can show that

$$\sum_{v \in N_a} \deg(v, H_1) + \sum_{i=1}^m \sum_{v \in M_{ai}} \deg(v, H_{2i}) \leq 2(N_a + \sum_{i=1}^m M_{ai}). \quad (20)$$

We now claim that any vertex v in H_1 that represents an inactive component in N_d must have its degree $\deg(v, H_1) \geq 2$ unless the inactive component contains the depot d_1 . This result follows from the fact that a component, which does not contain d_1 , can become inactive in \mathcal{C}_1 only if the constraint associated with this component in (12) becomes tight. Therefore, all the vertices in this inactive component must be marked. Also, if vertex v is a leaf ($\deg(v, H_1) = 1$) then pruning all the edges from this inactive component will not disconnect any unmarked target from d_1 . Hence, the pruning step of the algorithm will ensure that an inactive component can never be a leaf vertex in H_1 unless it contains d_1 . Now, we show the final part of the proof:

$$\sum_{v \in N_a} \deg(v, H_1) + \sum_{i=1}^m \sum_{v \in M_{ai}} \deg(v, H_{2i}) \quad (21)$$

$$= \sum_{v \in N_a \cup N_d} \deg(v, H_1) - \sum_{v \in N_d} \deg(v, H_1) \quad (22)$$

$$+ \sum_{i=1}^m \left[\sum_{v \in M_{ai} \cup \{M_d\}} \deg(v, H_{2i}) - \deg(M_d, H_{2i}) \right] \quad (23)$$

$$\leq 2(|N_a| + |N_d| - 1) - (2|N_d| - 1) + 2 \sum_{i=1}^m |M_{ai}| \quad (24)$$

$$< 2|N_a| + 2 \sum_{i=1}^m |M_{ai}|. \quad (25)$$

Hence proved.

IV. APPROXIMATION ALGORITHM FOR THE 2DHTSP

The approximation algorithm for the 2DHTSP is shown in algorithm 4.

Algorithm 4 : Approximation algorithm for 2DHTSP

- 1: Compute a heterogeneous spanning forest using the primal-dual algorithm. Let the two trees corresponding to the first and the second vehicle be denoted as $Tree_1$ and $Tree_2$ respectively.
 - 2: For $i = 1, 2$, do the following:
 - Form an Eulerian graph, \mathfrak{E}_i , for the i^{th} vehicle by doubling the edges in $Tree_i$.
 - Using \mathfrak{E}_i , find a walk that visits each of the edges in \mathfrak{E}_i exactly once. Next, shortcut this walk to find a tour for the i^{th} vehicle such that each vertex in \mathfrak{E}_i is visited exactly once.
-

Corollary 2: Algorithm 4 presented for 2DHTSP has an approximation ratio of 2.

Proof: Since the costs satisfy the triangle inequality, the sum of the cost of the tours obtained from the algorithm is at most equal to twice the cost of the edges in the trees. Therefore, from theorem III.1, it follows that the cost of

the feasible solution obtained is at most equal to twice the optimal cost of the 2DHTSP. In addition, the computational complexity of the algorithm 4 is dominated by the primal-dual procedure which can be implemented in $|T|^2 \log |T|$ steps. Therefore, the approximation ratio of algorithm 4 is 2. Hence proved. ■

REFERENCES

- [1] J. E. Davis M. Holland G. L. Feithans, A. J. Rowe and L. Berger, “Vigilant spirit control station (vscs)the face of counter,” in *Proc. AIAA Guidance, Navigation and Control Conf. Exhibition*. 2008, AIAA.
- [2] Jae-Ha Lee, Otfried Cheong, Woo-Cheol Kwon, Sung Yong Shin, and Kyung-Yong Chwa, “Approximation of curvature-constrained shortest paths through a sequence of points,” in *Proceedings of the 8th Annual European Symposium on Algorithms*, London, UK, 2000, ESA ’00, pp. 314–325, Springer-Verlag.
- [3] S. Rathinam, R. Sengupta, and S. Darbha, “A resource allocation algorithm for multivehicle systems with nonholonomic constraints,” *Automation Science and Engineering, IEEE Transactions on*, vol. 4, no. 1, pp. 98 –104, 2007.
- [4] W. Malik, S. Rathinam, and S. Darbha, “An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem,” *Operations Research Letters*, vol. 35, no. 6, pp. 747 – 753, 2007.
- [5] S. Rathinam and R. Sengupta, “3/2-approximation algorithm for two variants of a 2-depot hamiltonian path problem,” *Operations Research Letters*, vol. 38, no. 1, pp. 63 – 68, 2010.
- [6] Zhou Xu and Brian Rodrigues, “A 3/2-approximation algorithm for multiple depot multiple traveling salesman problem,” in *Algorithm Theory - SWAT 2010*, Haim Kaplan, Ed., vol. 6139 of *Lecture Notes in Computer Science*, pp. 127–138. Springer Berlin / Heidelberg.
- [7] Sai Yadlapalli, Sivakumar Rathinam, and Swaroop Darbha, “3-approximation algorithm for a two depot, heterogeneous traveling salesman problem,” *Optimization Letters*, pp. 1–12, 2010, 10.1007/s11590-010-0256-0.
- [8] Nicos Christofides, “Worst-case analysis of a new heuristic for the travelling salesman problem,” Tech. Rep., Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1976.
- [9] Vijay V. Vazirani, *Approximation Algorithms*, Springer-Verlag, Berlin, 2001.
- [10] Michel X. Goemans and David P. Williamson, “A general approximation technique for constrained forest problems,” *SIAM J. Comput.*, vol. 24, no. 2, pp. 296–317, 1995.